# *Synchronization Part 1*

*REK's adaptation of Claypool's adaptation of*

*Tanenbaum's*

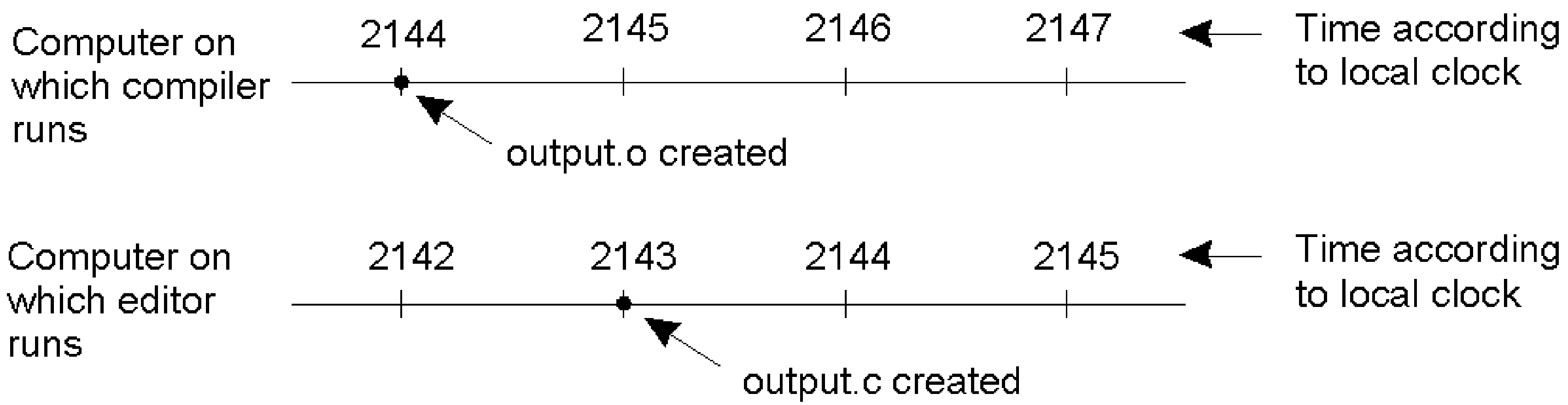
*Distributed Systems*

*Chapter 5*

# Outline

- *Clock Synchronization*
- *Clock Synchronization Algorithms*
- *Logical Clocks*
- *Election Algorithms*
- *Mutual Exclusion*
- *Distributed Transactions*
- *Concurrency Control*

# Clock Synchronization
## *make* example



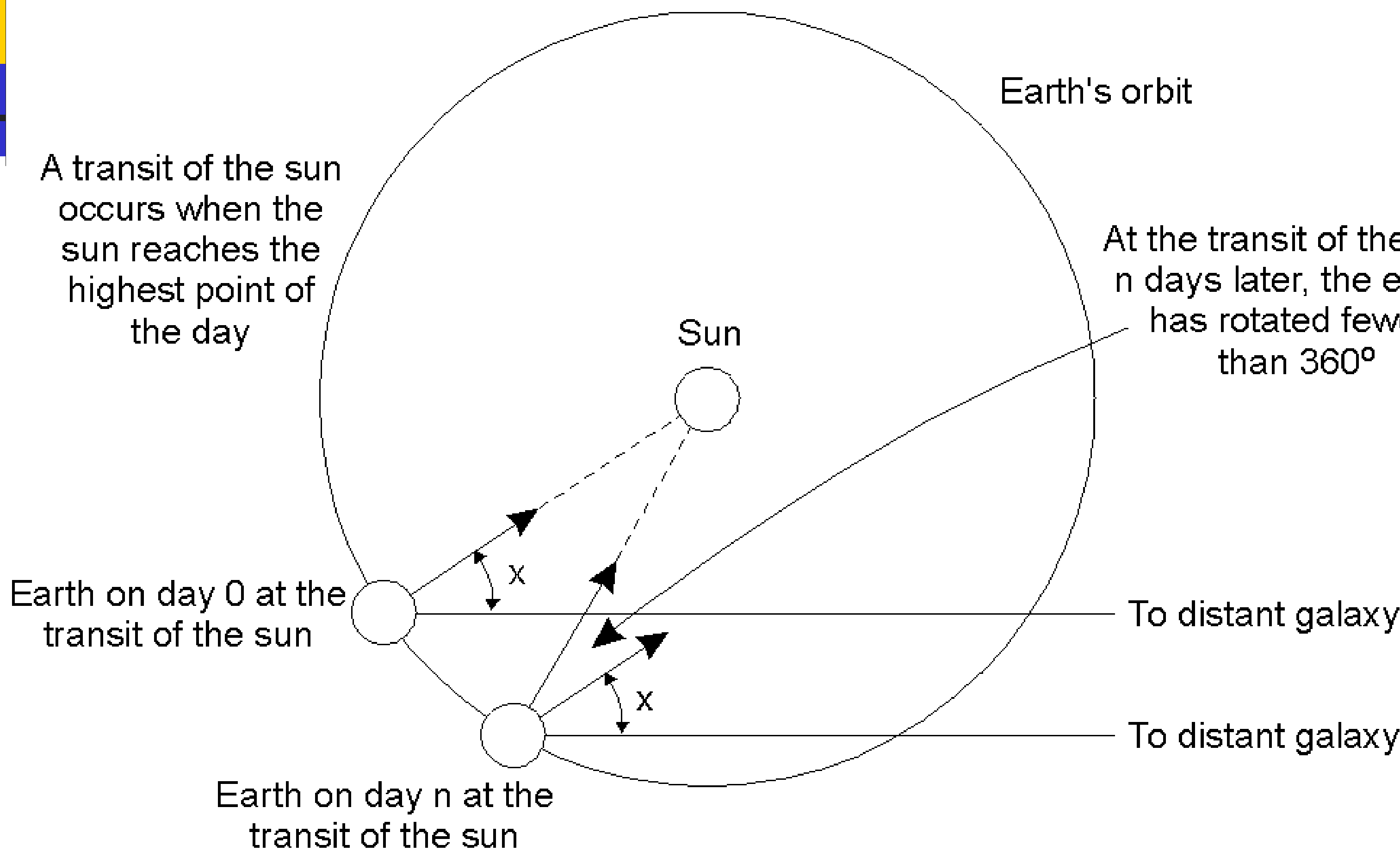

- *When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.*

- Same holds when using NFS mount
- Can all clocks in a distributed system be synchronized?

# Physical Clocks

- *It is impossible to guarantee that crystals in different computers all run at exactly the same frequency. This difference in time values is clock skew.*
- *"Exact" time was computed by astronomers*
  - *The difference between two transits of the sun is termed a solar day. Divide a solar day by 24*60*60 yields a solar second.*
- *However, the earth is slowing! (35 days less in a year over 300 million years)*
- *There are also short-term variations caused by turbulence deep in the earth's core.*
  - *A large number of days (n) were used used to the average day length, then dividing by 86,400 to determine the mean solar second.*

# Physical Clocks




*Computation of the mean solar day.*

# *Physical Clocks*

- *Physicists take over from astronomers and count the transitions of cesium 133 atom*
  - *9,192,631,770 cesium transitions == 1 solar second*
  - *50 International labs have cesium 133 clocks.*
  - *The Bureau Internationale de l'Heure (BIH) averages reported clock ticks to produce the International Atomic Time (TAI).*
  - *The TAI is mean number of ticks of cesium 133 clocks since midnight on January 1, 1958 divided by 9,192,631,770 .*

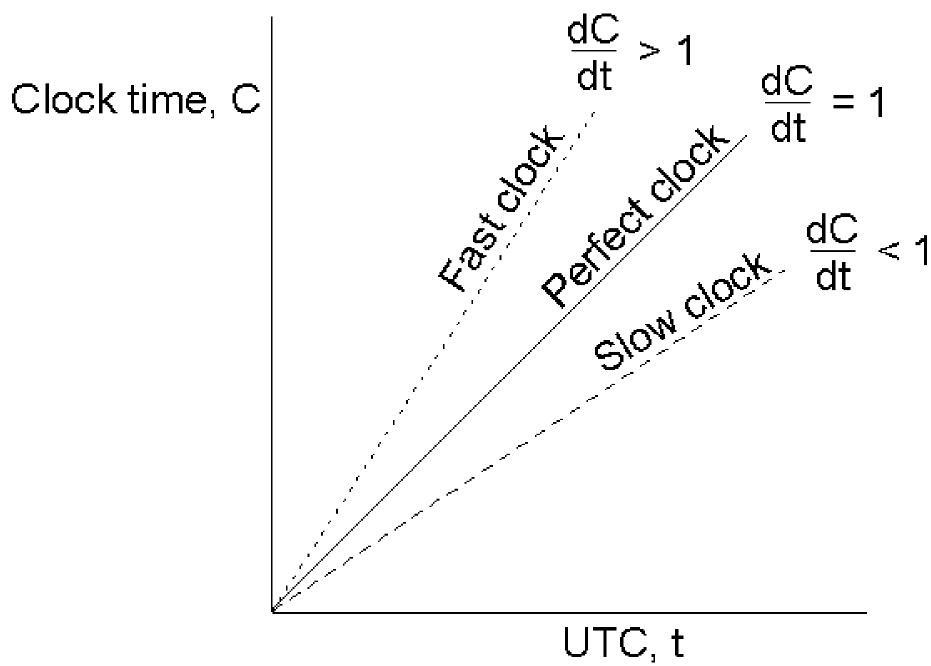
# Physical Clocks

- *To adjust for lengthening of mean solar day, leap seconds are used to translate TAI into Universal Coordinated Time (UTC).*
- *UTC is broadcast by NIST from Fort Collins, Colorado over shortwave radio station WWV. WWV broadcasts a short pulses at the start of each UTC second.* **[*accuracy 10 msec.*]**
- *GEOS (Geostationary Environment Operational Satellite) also offer UTC service.* **[*accuracy 0.5 msec.*]**

# Outline

- *Clock Synchronization*
- → *Clock Synchronization Algorithms*
- *Logical Clocks*
- *Election Algorithms*
- *Mutual Exclusion*
- *Distributed Transactions*
- *Concurrency Control*

# *Clock Synchronization Algorithms*




- Computer timers go off *H* times/sec, and increment the count of ticks (interrupts) since an agreed upon time in the past.
- This clock value is C.
- Using UTC time, the value of clock on machine p is $C_p(t)$.
- For a perfect time, $C_p(t) = t$ and $dC/dt = 1$.
- For an ideal timer, *H* =60, should generate 216,000 ticks per hour.

# *Clock Synchronization Algorithms*

- But typical errors, $10^{-5}$, so the range of ticks per second will vary from 215,998 to 216,002.
- Manufacturer specs can give you the *maximum drift rate* (ρ).
- Every $\Delta t$ seconds, the worst case drift between two clocks will be at most $2\rho\Delta t$.
- To guarantee two clocks never differ by more than $\delta$, the clocks must re-synchronize every $\delta/2\rho$ seconds using one of the various clock synchronization algorithms.

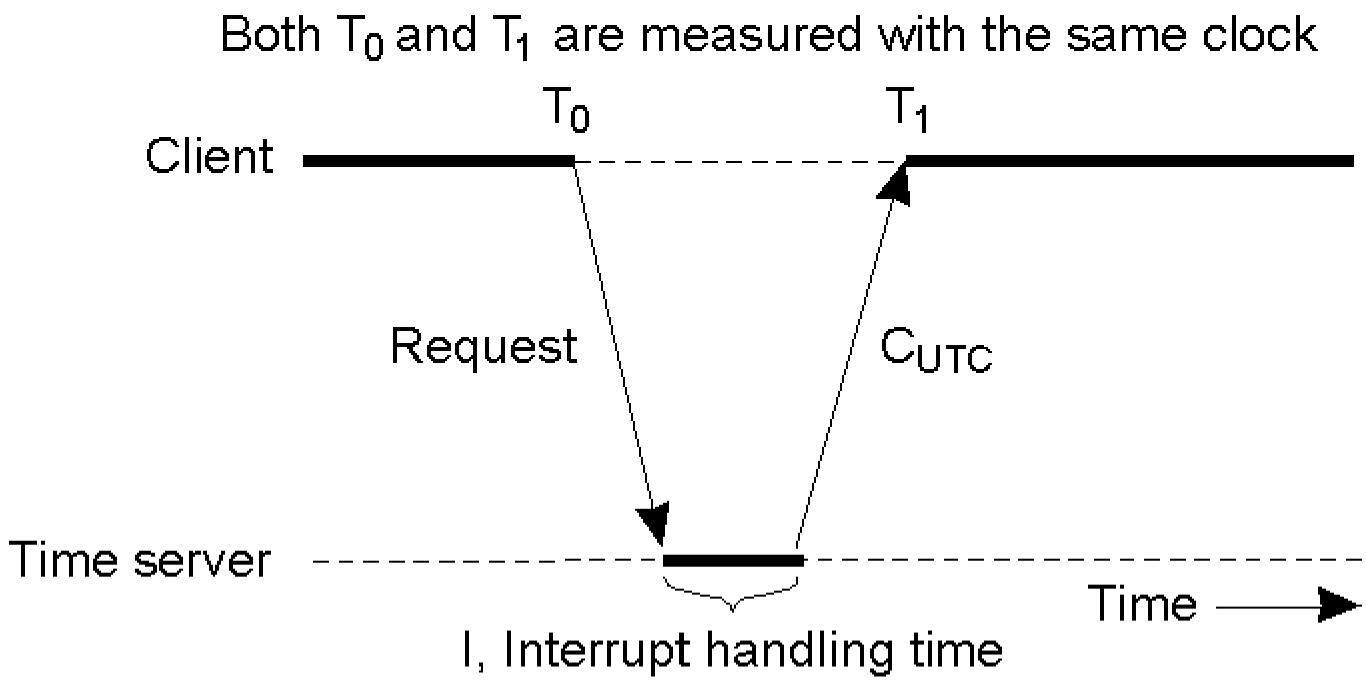
# *Clock Synchronization Algorithms*

- *Centralized Algorithms*
  - *Cristian's Algorithm (1989)*
  - *Berkeley Algorithm (1989)*
- *Decentralized Algorithms*
  - *Averaging Algorithms (e.g. NTP)*
  - *Multiple External Time Sources*

# Cristian's Algorithm

- *Assume one machine (the time server) has a WWV receiver and all other machines are to stay synchronized with it.*
- *Every $\delta/2\rho$ seconds, each machine sends a message to the time server asking for the current time.*
- *Time server responds with message containing current time, $C_{UTC}$.*

# Cristian's Algorithm




*Getting the current time from a time server*

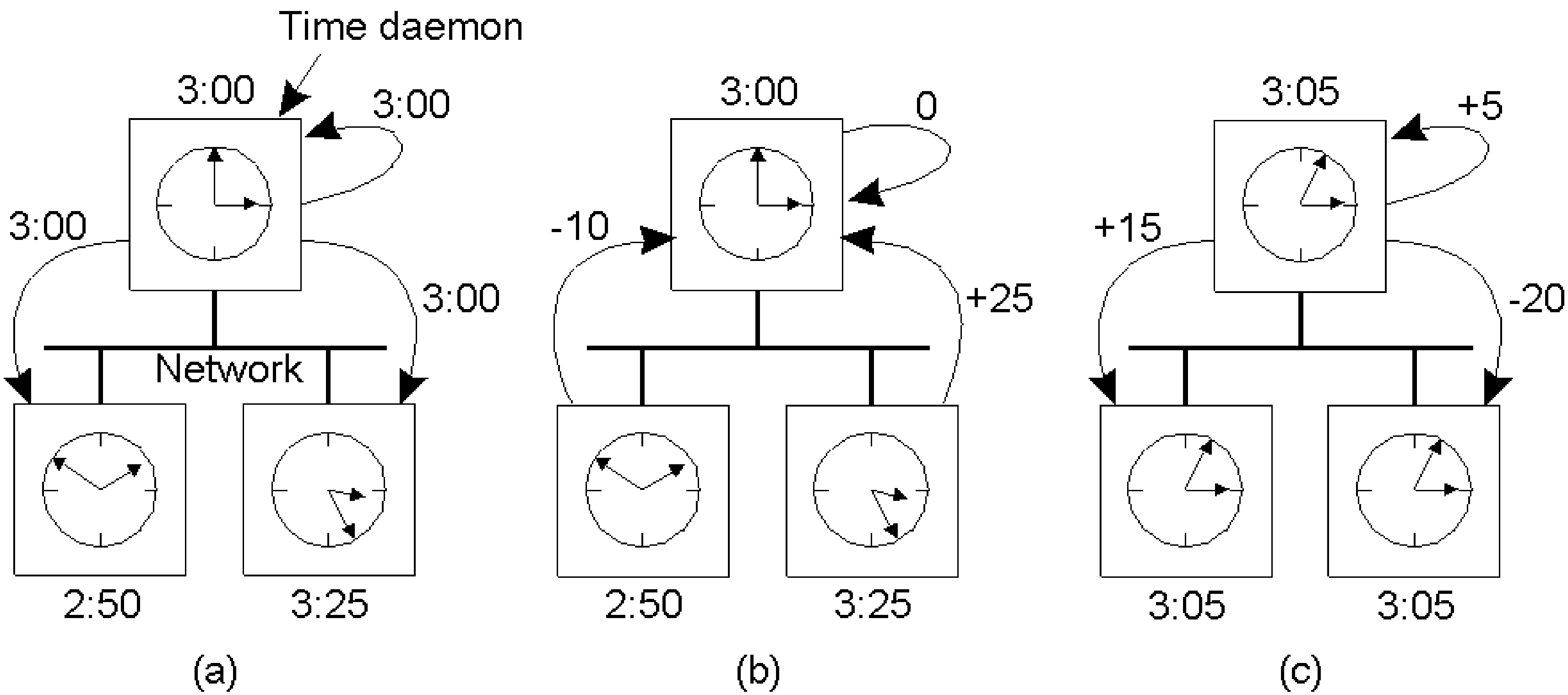
# Cristian's Algorithm

- *A major problem – the client clock is fast ➔ arriving value of $C_{UTC}$ will be smaller than client's current time, $C$.*
  - *What to do?*
    - ***One needs to gradually slow down client clock by adding less time per tick.***

# Cristian's Algorithm

- *Minor problem – the one-way delay from the server to client is "significant" and may vary considerably.*
  - *What to do?*
    - ***Measure this delay and add it to $C_{UTC}$.***
    - *The best estimate of delay is $(T_1 - T_0)/2$.*
  - *In cases when $T_1 - T_0$ is above a threshold, then ignore the measurement.* {outliers}
  - *Can subtract off I (the server interrupt handling time).*
  - *Can use average delay measurement or relative latency (shortest recorded delay).*

# The Berkeley Algorithm




a) The time daemon asks all the other machines for their clock values.
b) The machines answer and the time daemon computes the average.
c) The time daemon tells everyone how to adjust their clock.

# *Averaging Algorithms*

- *Every R seconds, each machine broadcasts its current time.*
- *The local machine collects all other broadcast time samples during some time interval, S.*
- *The simple algorithm:: the new local time is set as the average of the value received from all other machines.*

# *Averaging Algorithms*

- *A slightly more sophisticated algorithm :: Discard the m highest and m lowest to reduce the effect of a set of faulty clocks.*
- *Another improved algorithm :: Correct each message by adding to the received time an estimate of the propagation time from the $i^{th}$ source.*
  - *extra probe messages are needed to use this scheme.*
- *One of the most widely used algorithms in the Internet is the Network Time Protocol (NTP).*
  - *Achieves worldwide accuracy in the range of 1-50 msec.*

# *Outline*

- *Clock Synchronization*
- *Clock Synchronization Algorithms*
- *Logical Clocks*
- *Election Algorithms*
- *Mutual Exclusion*
- *Distributed Transactions*
- *Concurrency Control*

# *Logical Clocks*

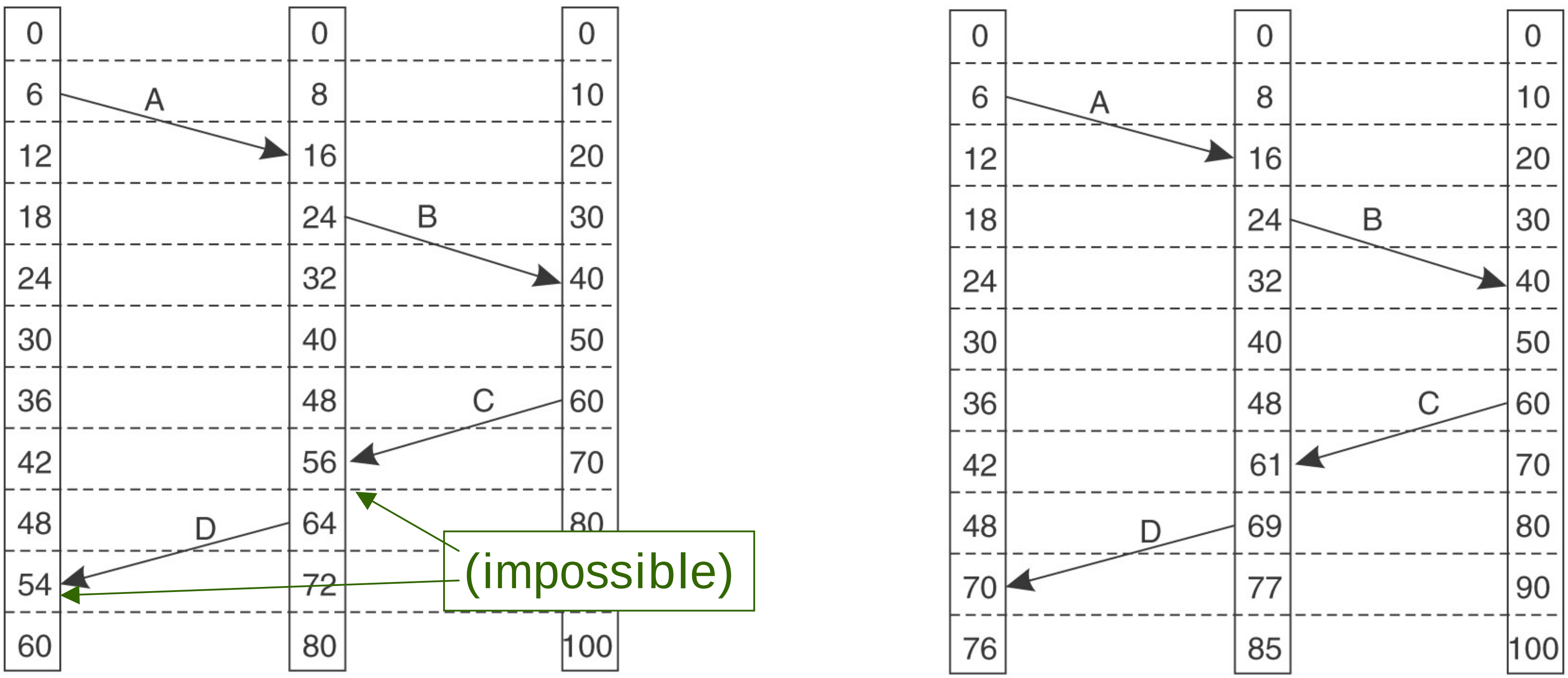
- *For a certain class of algorithms, it is the internal consistency of the clocks that matters. The convention in these algorithms is to speak of logical clocks.*
- *Lamport showed clock synchronization need not be absolute. What is important is that all processes agree on the order in which events occur.*

# *Lamport Timestamps [1978]*

- Lamport defined a relation "happens before". a → b *'a happens before b'.*
- Happens before is observable in two situations:

1. *If a and b are events in the same process, and a occurs before b, then a → b is true.*
2. *If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then a → b is also true.*

# Lamport Timestamps



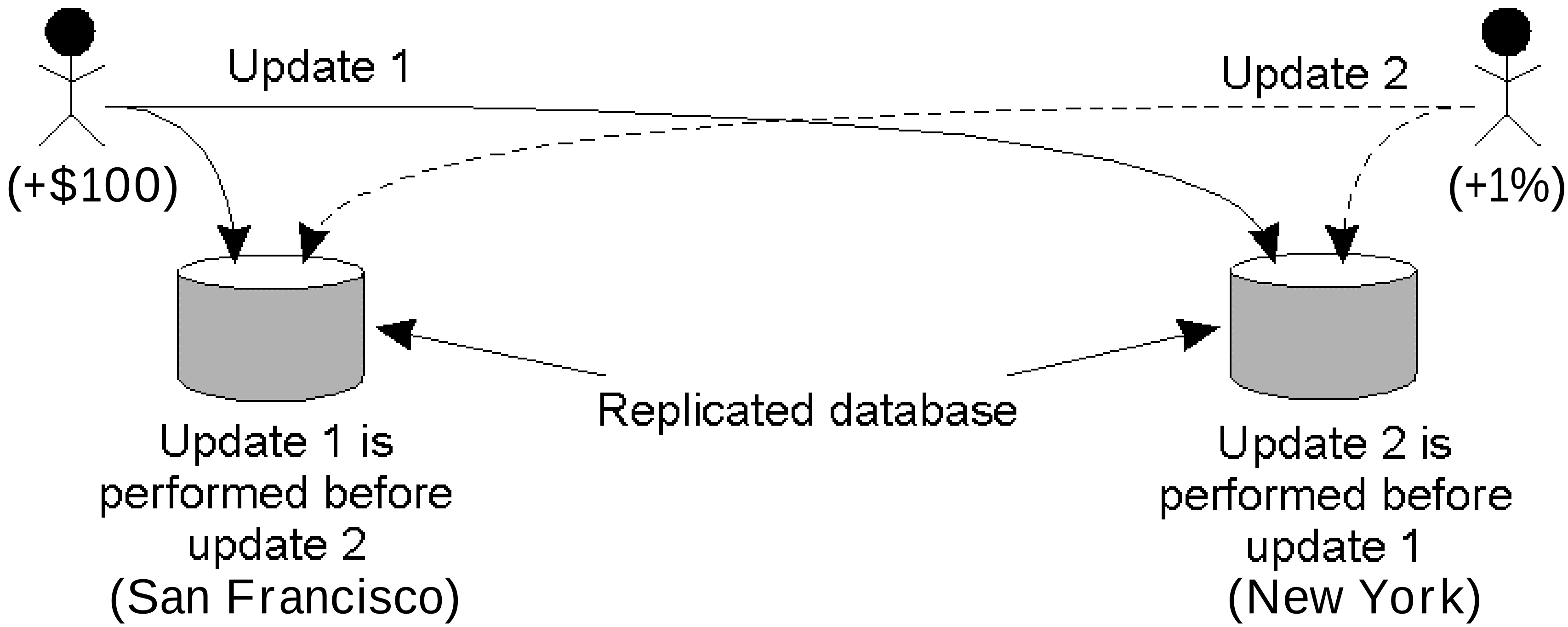

(a) (b)

*a) Each processes with own clock with different rates.*

*b) Lamport's algorithm corrects the clocks.*

*c) Can add machine ID to break ties*

# Example: Totally-Ordered Multicasting




- *San Fran customer adds $100, NY bank adds 1% interest*
  - *San Fran will have $1,111 and NY will have $1,110*
- *Updating a replicated database and leaving it in an inconsistent state.*
- *Can use Lamport's to totally order*

# Totally-Ordered Multicast

- A multicast operation by which all messages are delivered in the same order to each receiver.
- Lamport Details:
  - Each message is timestamped with the current logical time of its sender.
  - Multicast messages are conceptually sent to the sender.
  - Assume all messages sent by one sender are received in the order they were sent and that no messages are lost.

# Totally-Ordered Multicast

- *Lamport Details (cont):*
  - *Receiving process puts a message into a local queue ordered according to timestamp.*
  - *The receiver multicasts an ACK to all other processes.*
  - *Key Point from Lamport: the timestamp of the received message is lower than the timestamp of the ACK.*
  - *All processes will eventually have the same copy of the local queue → consistent global ordering.*